matrix.py

```python
class Matrix:
    def __init__(self, _content : list[list[float]] | tuple[float], _is_vector : bool = False) -> None:
        if _is_vector:
            self.V = _content
            self.m = 1
            self.n = len(self.V)
        else:
            self.M = _content
            self.m = len(self.M)
            self.n = len(self.M[0])
        self.is_vector = _is_vector

    def to_vertex(self) -> tuple[float, float, float]:
        if self.is_vector:
            return self.V

    def mul(self, v1, v2):
        return sum([v1[i] * v2[i] for i in range(len(v1))])

    def T(self):
        content = [[0 for _ in range(self.m)] for _ in range(self.n)]
        for m in range(self.m):
            for n in range(self.n):
                content[n][m] = self.M[m][n]
        return Matrix(content)



    def __mul__(self, _mul):
        if self.is_vector and _mul.is_vector and self.n == _mul.n:
            return self.mul(self.V, _mul.V)
        elif not self.is_vector and _mul.is_vector:
            return Matrix(tuple([self.mul(row, _mul.V) for row in self.M]), True)
        elif not self.is_vector and not _mul.is_vector and self.n == _mul.m:
            content = []
            _mul_t = _mul.T()
            for line_a in self.M:
                row = []
                for line_b in _mul_t.M:
                    row.append(self.mul(line_a, line_b))
                content.append(row)
            return Matrix(content)


    def __repr__(self) -> str:
        if self.is_vector:
            return self.V.__repr__()
        else:
            ret = '[\n'
            for row in self.M:
                ret += f'    {row}\n'
            ret += ']'
            return ret


```

```
Identity3_Matrix = Matrix(
    [
        [1, 0, 0],
        [0, 1, 0],
        [0, 0, 1]
    ]
)

Zero3_Matrix = Matrix(
    [
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]
    ]
)

Zero3_Vector = Matrix((0, 0, 0), True)

from math import cos, sin

M = {
    'T'     : lambda _vector :  Matrix([
                                    [1, 0, 0, _vector[0]],
                                    [0, 1, 0, _vector[1]],
                                    [0, 0, 1, _vector[2]],
                                    [0, 0, 0, 1],
                                ]),
    'Ro_x'  : lambda _angle :   Matrix([
                                    [1, 0, 0],
                                    [0, cos(_angle), -sin(_angle)],
                                    [0, sin(_angle), cos(_angle)],
                                ]),
    'Ro_y'  : lambda _angle :   Matrix([

                                    [cos(_angle), 0, sin(_angle)],
                                    [0, 1, 0],
                                    [-sin(_angle), 0, cos(_angle)],
                                ]),
    'Ro_z'  : lambda _angle :   Matrix([
                                    [cos(_angle), -sin(_angle), 0],
                                    [sin(_angle), cos(_angle), 0],
                                    [0, 0, 1],
                                ]),
    'Sc'    : Zero3_Matrix,
    'Re_x'  : Zero3_Matrix,
    'Re_y'  : Zero3_Matrix,
    'Re_z'  : Zero3_Matrix,
}

def Ro_Any_Matrix(_yaw : float, _pitch : float, _roll : float) -> Matrix:
    return M['Ro_z'](_yaw) * M['Ro_y'](_pitch) * M['Ro_x'](_roll)

if __name__ == '__main__':
    print(Ro_Any_Matrix(1, 1, 1))
```